

CSC 400 Final Report

Donos

Marquis Lockhart & Paul Osorio

18th August, 2020

[GitHub Repository](#)



Table of Contents

1. Introduction.....	1
1.1. Background Information & Motivation.....	2
1.2. Project Objectives.....	2
2. Program Design & Implementation.....	3
2.1. User Interface.....	4
2.2. Program Design.....	6
3. Testing & Evaluation.....	11
4. State of Implementation.....	14
4.1. Features.....	14
4.1.1. Core Features.....	14
4.1.2. Secondary Features.....	14
4.2. Future Implementation.....	15
5. Reflection.....	17
5.1. Main Challenges.....	17
5.2. Acquired Skills.....	17
5.3. Resources.....	1
5.4. Final Thoughts.....	17

1. Introduction

1.1. Background Information & Motivation

In recent years, there's been a rise in the amount of websites that allow users to create a cause that others can then donate to (ex. GoFundMe, Kickstarter). They've proven to be successful and have continued to rise in popularity. Most certified organizations still seem to be using the traditional methods of running all of their operations on-ground and not using the internet to their potential. This may be because while GoFundMe or Kickstarter serves about the same purpose, most certified organizations accept more than just money. Well-renowned organizations such as Action Against Hunger or Toys for Tots can benefit from a way to receive donations for food, toys, or their respective requests in the same way they can receive money on the previously mentioned sites.

The initial idea for the project came about during the COVID-19 pandemic. Southern Connecticut State University sent out an email stating that they'd be giving out all food leftover on the campus prior to all students, faculty, and staff leaving the campus indefinitely. The email seemed to be easily overlooked and buried under the rest of the emails regarding the pandemic, so it's likely a majority of that food still went to waste. We thought there needed to be a way to bring more awareness to these sort of charities and drives when they take place.

With this in mind, our motivation for the project was to create an application that allows for non-profit organizations and charities to create online drives where they can receive donations from donors, while spreading the awareness and existence of these charities.

1.2. Project Objectives

Donos is an application that displays a local community feed of nearby donation drives and local charities. Our application aims to raise awareness and build community interactions by notifying users where they can donate or receive charitable donations by their community members.

To do this, our project needed to have two different user types: organizations and community members(referred to as users). Two different classes were needed because the respective user types should only have specific views and functions available to them. Organizations should have the ability to create charity drives and receive donations towards them. Users should be able to browse, follow, and donate to those created charity drives, and discover local charities in their communities.

2. Program Design & Implementation

For this program we worked with Agile software development, which approaches the development process by breaking it up into microcycles. The work of the project was broken down into smaller sprints, allowing us to have more specific deadlines on the functions we wanted to implement and narrowing our focuses. The Agile development lifecycle includes the following steps:

1. **Specifying Requirements** - In this step, projects are envisioned and prioritized. This was completed upon defining our project motivation and objectives.
2. **Incepting the Design** - Since we already knew our team, this step was more focused around figuring out what technologies would be used, as well as the requirements needed to complete the project objectives.
3. **Iterating the Development** - Our biweekly meetings with the instructor of the course along with our weekly, sometimes daily, meetings with one another allowed us to adjust the software being developed using feedback provided.
4. **Testing** - Quality assurance testing was ongoing throughout the course, with a week dedicated to that alone towards the end of the production life cycle.
5. **Deployment** - Following the first completed version of our application for presentation, we deployed the website to a live server for ease of access and usage.
6. **Review** - This step would allow for us to make adjustments to our application, similar to the third step. After the review, the cycle would loop and continue.



Figure 2.1: The Agile software development life cycle.

2.1. User Interface

The design of our front-end was more complex than we originally expected. This can possibly be because of the guidelines we laid out for the front-end design. Taking our knowledge from another course we've both previously taken, Human Computer Interaction (CSC 334), we wanted to implement many of the practices we learned about user interface design and user experience. Our goal for the website was a design that was user-friendly and easy to use.

To do this, we referred to Jakob's Law which states "Users spend most of their time on other sites. This means that users prefer your site to work the same way as all the other sites they already know" (<https://lawsofux.com/jakobs-law>). To meet this requirement, there were five key principles of website design that we followed: availability/accessibility, clarity, learnability, credibility, and relevancy.

With the recent high success of charity and funding sites such as GoFundMe or Kickstarter, we thought our application should relate to their structure and functionality to take advantage of relevancy. With this inspiration in mind, we replicated a drive list on the home page where you can filter by specific views. We decided to make our filters by drives hosted in the user's city or state in an attempt to promote the objective of local charities. The inspiration from modern websites helped us make sure the interface and experience we were creating was relevant.

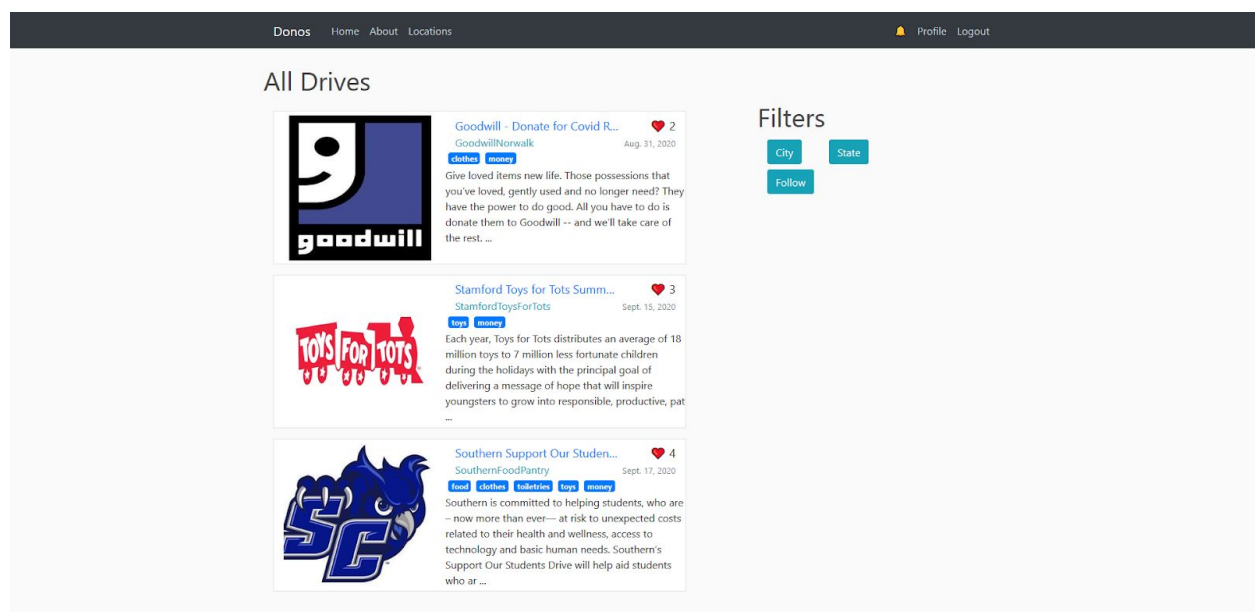


Figure 2.2: List view of drives on the home page.

To ensure credibility, we wanted the website to look professional. Our way of doing this was adding a banner image to drive and organization pages, replicating what you see on most social media today. Showing images as opposed to just text improves the credibility of not only our website, but the organizations themselves. Another small feature added for extra credibility was the “verified” mark next to an organization’s name on the organization page. This also references the blue checks symbolizing authenticity that many social media sites use.

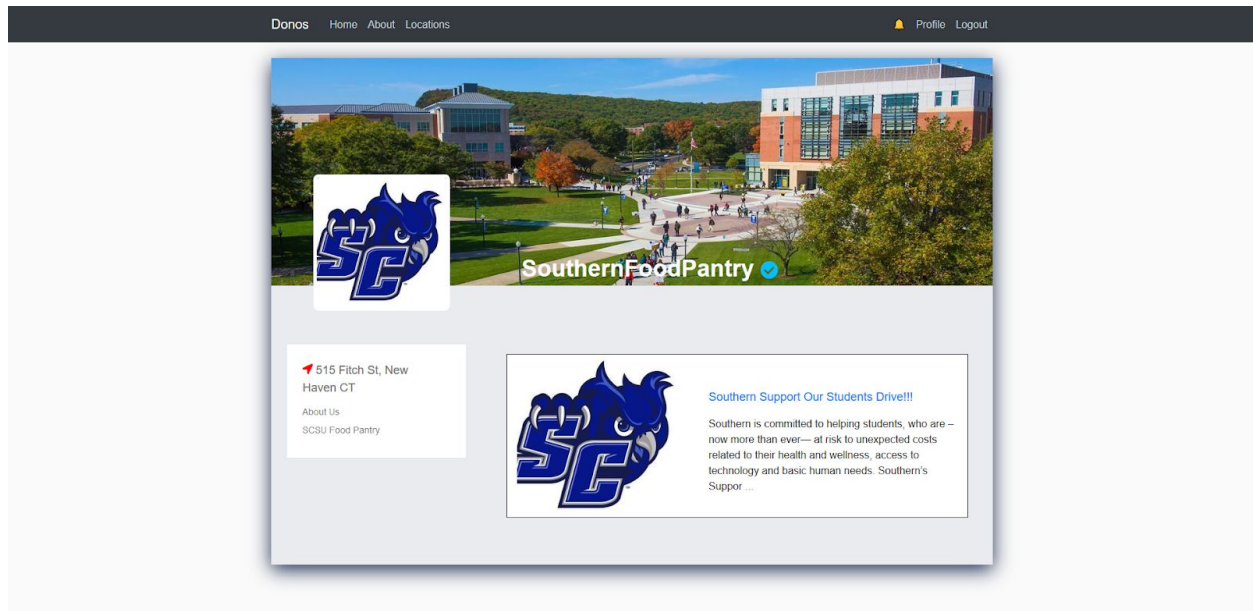


Figure 2.3: Organization view.

Accessibility and clarity went hand in hand with this application. We wanted everything to be straightforward for the different types of user views since there were a good amount. It would be clear for an admin that they were one since they’d mostly be viewing the site from the admin panel. For organizations and base users, their views are very similar overall with few differences that can be subtle. To better differentiate between the two, we implemented a different color navigation bar based on your user class: blue for organizations and black for base users. Although this may be something either user will never see, it’s a nice addition from a developer standpoint to amplify the clarity and accessibility of the interface.

To further promote the objective of Jakob’s Law into the user interface, we’ve created a visual map on the website to view local charities, non-profit organizations and soup kitchens. We decided that we wanted to have a list of them locally so it’s easy to brief through them from an experience standpoint, but a map as well since it’s much more appealing aesthetically from an interface standpoint.

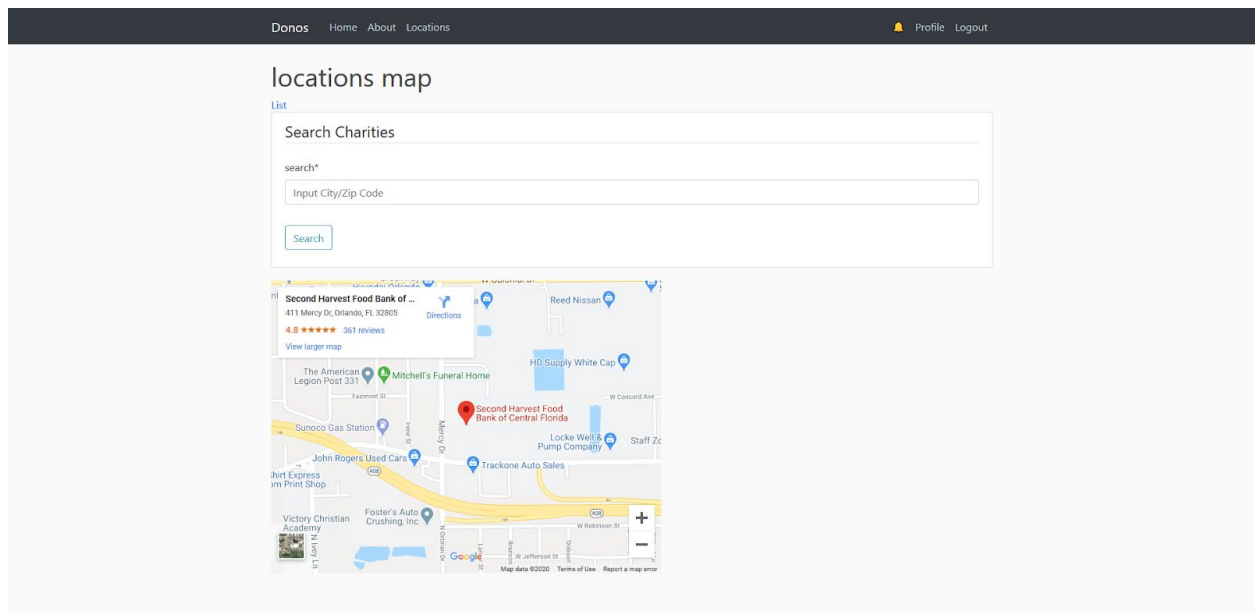


Figure 2.4: Map for users to find local charities.

2.2. Program Design

The design of the program can be broken down into its user classes and their specific functionalities as well as the incorporated database structure. Figure ## shows a breakdown of how the users of the application would interact with one another through the features implemented. Figure ##(Database) shows the relationships between each table and what values are stored inside of each field.

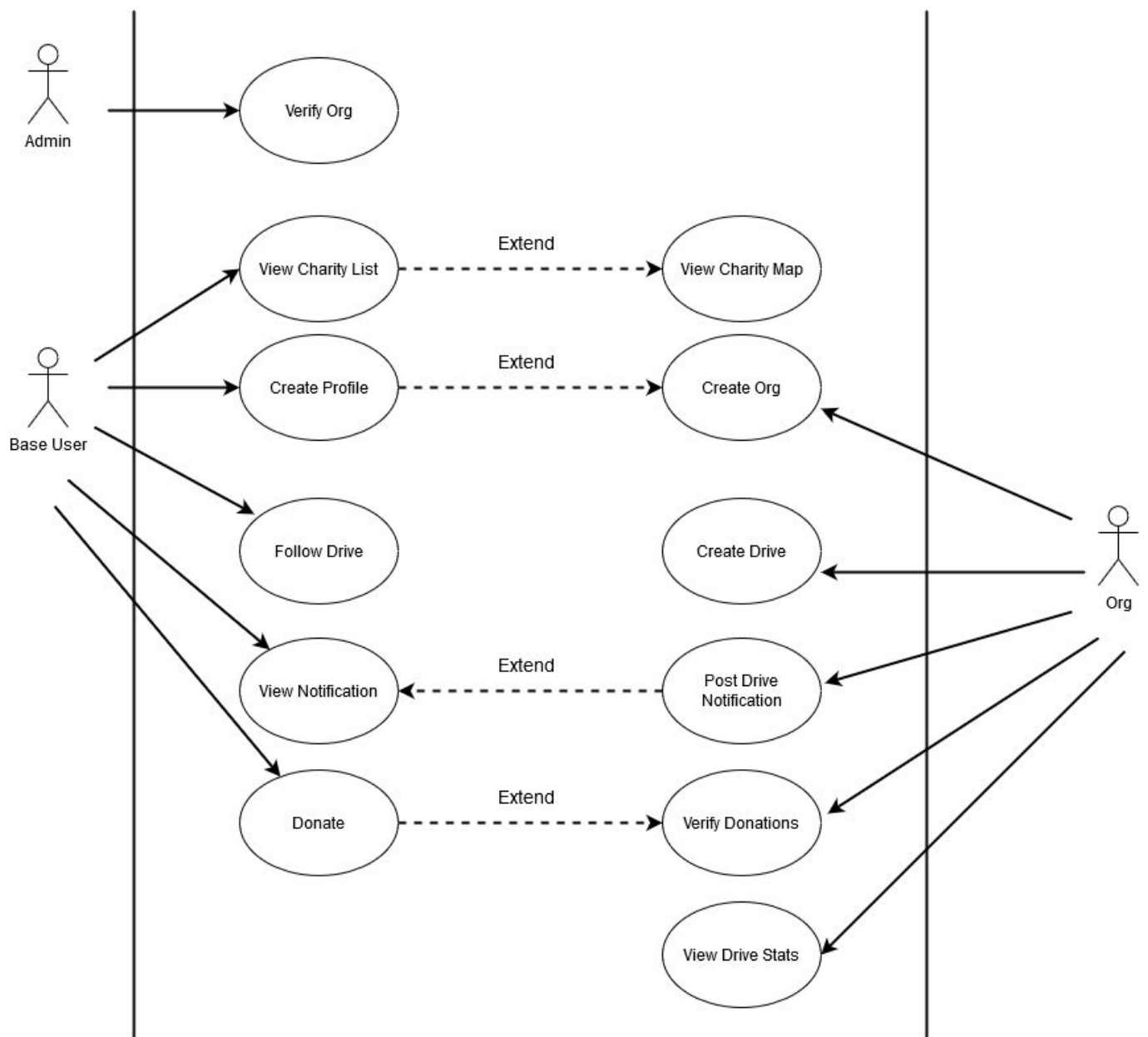


Figure 2.5: Conceptual diagram for use cases.

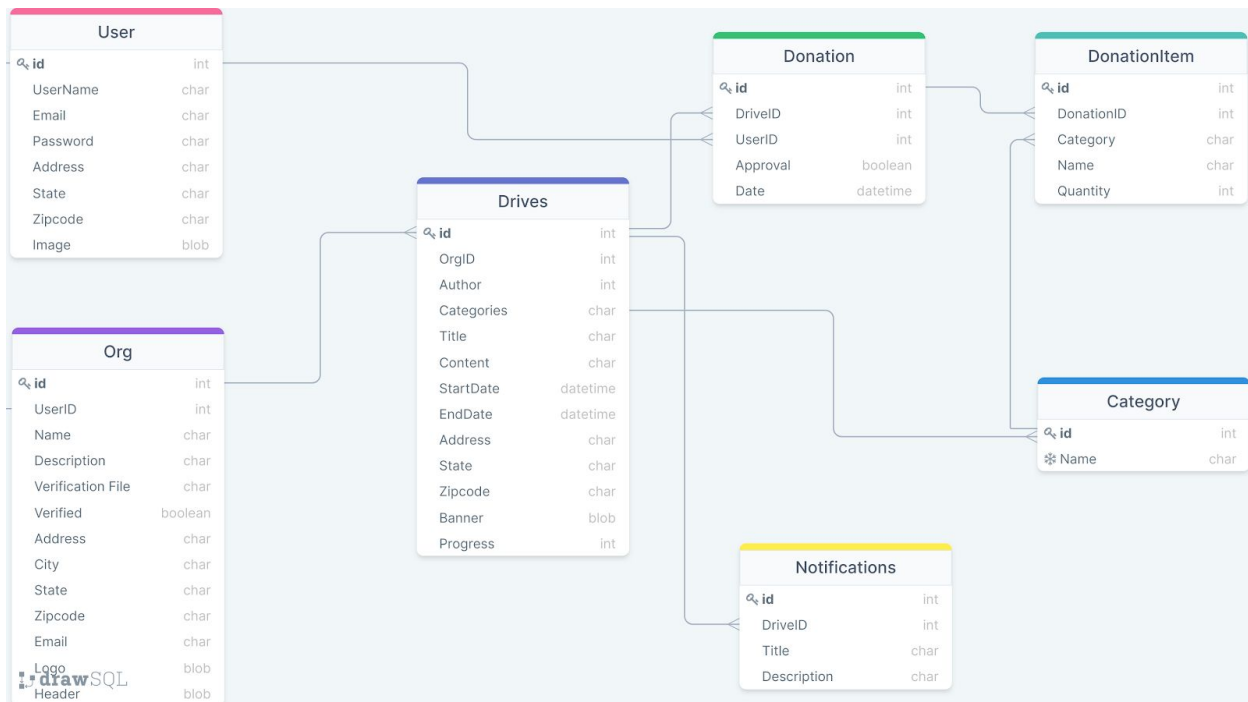


Figure 2.6: Entity Relationship Diagram for database structure.

In the source code itself, there are many views that allow the user functions to operate. In the figures below, we showcase and explain the backend of some of the important features that were instrumental to our application.

```

class DriveListView(ListView):
    model = Drive
    template_name = 'donos/home.html'
    context_object_name = 'drives'
    paginate_by = 5

    def get_queryset(self):
        return Drive.objects.filter(end_date__gt=timezone.now()).order_by('-start_date')
  
```

Figure 2.7: Class based view of the home page.

In Figure 2.7, we decided to use class based views to display our drives in a list pattern because they provided us with more tools to further extend and customize compared to a function based view. The function `get_queryset` mutates the queryset to display not all drives but instead drives that have not expired and are ordered by descending start dates.

```

@login_required()
def donate(request, pk, fnum):
    formset_base = formset_factory(DonationForm, extra=fnum)
    # WITHOUT form_kwargs CUSTOM PARAMETERS IN FORMS WOULD NOT WORK
    formset = formset_base(form_kwargs={'id': pk})
    helper = DonationFormSetHelper()

    # Validate Donation Form
    if 'submit' in request.POST:
        formset = formset_base(request.POST, form_kwargs={'id': pk})
        if formset.is_valid():
            # creates donation object
            d = Donation.objects.create(drive=Drive.objects.get(id=pk), user=request.user, approved=False)
            d.save()

            # retrieves donation object from database
            d.refresh_from_db()
            for form in formset:
                # Assigns each DonationItem object to Current Donation
                obj = form.save(commit=False)
                obj.donation = d
                obj.save()
            return redirect('drive-detail', pk)
        else:
            messages.error(request, "Not valid!")
    # Increments DonationItem form
    elif 'form_add' in request.POST:
        fnum = fnum + 1
        return redirect('drive-donate', pk, fnum)
    # Decrements Donation Item form
    elif 'form_remove' in request.POST:
        fnum = fnum - 1
        return redirect('drive-donate', pk, fnum)

```

Figure 2.8: Function based view of user donation submission.

In figure 2.8, we created a Donation formset of a list of DonationItem forms. We had 3 Post requests. Submit to submit the data to the database. Form_add to increment DonationItem forms. Form_remove to decrement DonationItem forms.

To submit a Donation Form with its respective DonationItems we first had to create a Donation object and commit it to the database before validating the DonationItems. This required using the *refresh_from_db* function to commit the Donation object and retrieve its primary key, preventing a foreign key restraint error.

```
@login_required
def announcements_drives(request):
    user = request.user
    drives = user.profile.follows.all()

    page = request.GET.get('page')

    # create an empty queryset
    q = Notifications.objects.none()

    # union multiple queries together
    for drive in drives:
        q = q.union(drive.notifications_set.all())

    q = Paginator(q.order_by('-date_posted'), 5)
    context = {
        'data': q.get_page(page),
    }
    return render(request, 'users/announcements_drives.html', context=context)
```

Figure 2.9: Function based view of user's drive announcements.

In Figure 2.9, we wanted to display all announcements from drives the user follows, displayed in a descending order by date posted. To display all announcements in a descending order instead of by drive then date_posted we had to union all query sets of each drive's notifications.

```

def locations_map(request):
    link = 'https://www.google.com/maps/embed/v1/search?key={}&q={}'
    api_key = os.getenv('api_key')

    # Optimal search string
    text_query = 'charity OR food bank in {}'

    if request.method == 'POST':
        form = SearchForm(request.POST)
        if form.is_valid():
            # retrieves data from form field
            data = form.cleaned_data['search']
            query = text_query.format(data)
            # replaces spaces with + for api compatibility
            query = query.replace(" ", "+")

            link = link.format(api_key, query)
        else:
            # default location is user's zipcode
            query = text_query.format(request.user.profile.zipcode)
            query = query.replace(" ", "+")

            link = link.format(api_key, query)
            form = SearchForm()

```

Figure 2.10: Function based view of charity location map.

In figure 2.10 we used the google places api to retrieve an api call of 20 or less charitable organizations at a given location. We found “charity OR food bank” search query to retrieve the most accurate results of a wide range of charitable organizations.

3. Testing & Evaluation

For the entire project, we tested the code through our remote-to-local branches on a localhost since we didn’t have enough time to deploy the application to a live web server. Earlier in the project, one of us ran into an issue where PyCharm wouldn’t allow us to run our Django application created through a cloned repository due to errors we couldn’t solve. This led to us

having to fork the repository and work on two different ones. Our testing consisted of us screen sharing to program simultaneously and pushing our branches. After pushing our branches to their respective repositories, we would then have to merge these branches across forks to make sure we were always up to date on the same version of the code. It wasn't the most efficient way to test and update the website, but it improved on our Git skills.

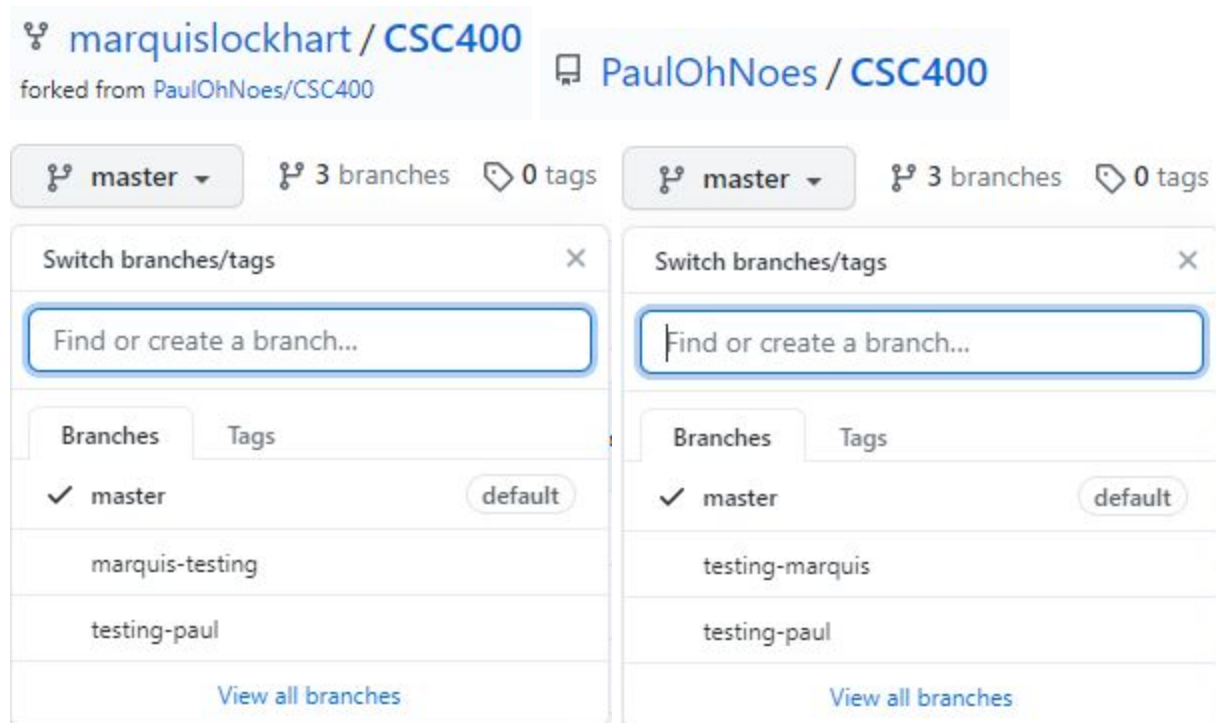


Figure 3.1: Repositories and branches for Marquis (left) and Paul(right).

Referring back to our Agile development process, we used our sprints to and current progress to let us know what should be in testing at the moment. For our initial project proposal, we created a Gantt chart that showed what we'd like to have in testing and when.

- **Sprint 1** (6/1 - 6/30): Build a skeleton of the Django framework and implement all main features (Donation Drive, Local Charities Map, User login) of the application.
- **Sprint 2** (7/1 - 7/31): Implementing the secondary components into the application and backend features (statistics, announcements page, donation submission form, payment processor).

- **Sprint 3 (8/1 - 8/17):** Field testing, optimization, and final polish of application.

Sprint 1				Sprint 2				Sprint 3		
6/1 - 6/8	6/9 - 6/16	6/17 - 6/24	6/25 - 6/30	7/1 - 7/8	7/9 - 7/16	7/17 - 7/24	7/25 - 7/31	8/1 - 8/8	8/9 - 8/16	Aug. 17th
SRS Part 1	Django Skeleton									
		Database								
		Core Feature Implementation (Charity Feed & Locations)								
				Backend Features (Login, Statistics)						
						Frontend (Visual Map, Announcements)				
								Domain/SSL Certification		
								Polish/Optimization		
										Final Presentation

Figure 3.2: Gantt chart timeline for project life cycle.

We tested through the entire project to make sure everything we were currently working on was completed, but towards the end, we dedicated an entire two week sprint to debugging and polishing the application. This consisted of finding ways that the user can potentially break the application and patching it. For example, if a base user tried to access a page only an organization should be viewing through the URL, they'd be met with a 403 error. This extra security in our application allows it to function with more integrity so that all users can feel safe using it. Although we were developing through a localhost and didn't get the chance to have test users, we developed the application to a point that it can realistically be used on a larger scale.

4. State of Implementation

4.1. Features

When coming up with our features, we had to prioritize what was most important in relation to the project objectives. This was relatively simple to do since we were working with an agile development process. We broke the features down into primary and secondary categories, pushing the core first and leaving the secondary toward the end of the development cycle.

4.1.1. Core Features

We defined our core features as those that are the heart of the application, meaning there would be no functionality without them.


One of the core features was the ability to create and follow donation drives. Verified organizations should be able to create a drive on the website, detailing a title, location, start and end date, description, and more. Once these drives are created, base users should be able to view them as a list on the home page, and follow the ones that they wanted to keep up with.

Going hand in hand with the previous feature, another core function was the ability to submit donations. When creating a drive, organizations are allowed to specify which categories of items they'd like to receive: *Food, Clothing, Toiletries, Toys, and/or Money*. Base users can choose to donate items to a drive of their choice, based on the specific requests that the organization was taking. Upon donating, the user would receive a donation number as confirmation and wait for the approval of the organization to ensure the submitted donations were actually received.

The final core feature was allowing base users to find local charities using a map. This specific feature focused on the objective of raising awareness and building community interactions. When accessing the map page, users will be shown a visual of all charities, non-profits, and soup kitchens that are local to their home. This is completed using the Google Maps API, so brief information is provided on it through the application. The user can access more information or directions for a given charity by using the external links provided to Google's website.

4.1.2. Secondary Features

Secondary features were defined as those that were equally as important as the core features, but they couldn't stand on their own. In other words, they relied on core features to be implemented first so that they can then be implemented after.



The first secondary feature that was implemented was the Announcements Page and Notifications Page. This relies on the core feature of creating drives. Once an organization has made a drive and posted it in full detail, they can start receiving donations. As the drive continues, they can create a post on the drive page that lets them communicate with the base users that follow. On the user's end, they can access their Notifications Page through the navigation bar. From here, they'll be able to see all announcement posts made from all of the drives that they follow.


The ability to approve donations was also major, relying on both the feature of creating drives as well as submitting donations. As previously mentioned, the base user would await the approval of their submitted donation by an organization. On the organization's end, they'd be able to view the submitted donations from the respective drive page. From here, they'd see a list of them as well as if they've been approved or not. Organizations can also search for a specific donation number if the base user were to present it to them in person. After clicking on the donation number, the organization can then verify that all submitted donation items are present and update any discrepancies.

The user and drive statistics was a feature that was implemented a bit later into production. This feature shows data such as the amount of followers, total donations, progress of a drive, expiration of a drive, top donors to a drive, and much more. We implemented this feature as a way for organizations to keep track of the success of their drives. For users, it'd also let them know what drives they donated to the most and all donations that they've made.

Finally, verifying organizations wasn't a feature that necessarily relied on any of the previous core functions, but it was important to the integrity of the application. We wanted a process to make sure only legit, certified non-profit organizations can make drives to prevent a random base user who only claims to be an organization from doing the same. To do this, we created a feature where the admin has to verify an organization through the admin panel prior to having access to drive creation. Upon registering as an organization, a file will have to be uploaded containing the legal documents showing that the organization is who they claim to be. They also register with an email at the same time, so if there isn't enough proof, the admin can reach out to the organization to request more information.

4.2. Future Implementation

At the time of our project presentation, all of the original features we planned to incorporate into the application were complete, so all future implementation would be additional things we didn't get to due to time constraints.



After discussing the outcome of the project, we both agreed that in the future we would want to work with the technologies we originally intended to but never got around to. One of those technologies was *React.js*. The first $\frac{3}{4}$ of the project's life cycle was mainly focused on implementing all of the backend and the final stretch focused on making the front end looking more presentable for the demo. Using React would've taken our project to the next level for the modern, familiar feel we were going for with the UI/UX, further bringing the website to life. It's also a technology neither of us have used before so it would've been nice to get a new tool in our skill set while sharpening our JavaScript skills.

Another thing we didn't get around to was deploying and hosting the application on a live server. This was something that we really wanted to do since we never got to do it for the projects we've previously worked on together in Software Design & Development (CSC 330) and Database Systems (CSC 335). We think having a hosted website as opposed to using a localhost again would've been a good chance to learn about scalability. However, it would've taken quite a bit of time to figure out especially since neither of us have previous experience with it.

Outside of new technologies, there's many features we came up with as production continued that we'd like to implement in the future. Some of these features include a payment processing system for base users to donate money to organizations and drives directly through the website. Another feature would be allowing users to make comments on the drives upon donating, and these comments would show on the drive page upon approval. We realized there's always going to be more that we want to do to the application, but we more than likely won't be able to make it happen because of our future endeavors taking priority. If there were people willing to work on the code and update the current version of the application to its full potential, we both agreed that's something we'd like to see in the future.

5. Reflection

5.1. Main Challenges

Our biggest challenge was understanding the scale of the project. Going into it, we were very ambitious with what we wanted to create, but due to time constraints and a lack of experience in some project development areas, we fell a bit short. We were both well versed in Flask coming into the course, so we wanted to challenge ourselves with a new framework -- Django. It was a lot more rigid in its structure than we anticipated, so it took us a majority of the course to learn it.

Another challenge was dividing the project tasks and features into sprints for 2 people. At times we overestimated our capabilities and therefore our backlog increased leaving us less time for next sprint tasks. Moving further we have a better understanding of how much time we should be allocating in different phases of software development.

5.2. Acquired Skills

As an outcome of the project, we've acquired a variety of skills and improved on some of our previous skills. Seeing that it was our main challenge, we had to become comfortable with Django and Git. After fully developing the project, we can both say that we're confident in our ability to work with these technologies at an industry level, but know there's much more for us to learn.

5.3. Resources

[Django Documentation](#)

[Google Places API](#)


[Bootstrap4](#)

[Corey Schafer Django Tutorials](#)

[Stackoverflow](#)

5.4. Final Thoughts

We are extremely happy with the outcome of the project. We've not only learned a lot during development but we also had a great time doing it as well. Part of this can be contributed to the chemistry we had as partners and the student-teacher relationship we held with our professor.



We'd like to see more progress on the project whether it's through our development or somebody else's because we know the potential it holds as a real world application. One way or another, we can say that we're ready to take the skills we've learned here and expand on them in the industry.

